McTiny:
McEliece for tiny network servers

Daniel J. Bernstein,
uic.edu, rub.de

Tanja Lange, tue.nl

Fundamental literature:
1962 Prange (attack)
+ many more attack papers.
1968 Berlekamp (decoder).
1970–1971 Goppa (codes).
1978 McEliece (cryptosystem).
1986 Niederreiter (compression)
+ many more optimizations.

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

McTiny:

McEliece for tiny network servers

Daniel J. Bernstein,

`uic.edu`, `rub.de`

Tanja Lange, `tue.nl`

Fundamental literature:

1962 Prange (attack)

+ many more attack papers.

1968 Berlekamp (decoder).

1970–1971 Goppa (codes).

1978 McEliece (cryptosystem).

1986 Niederreiter (compression)

+ many more optimizations.

Encoding and decoding

1978 McEliece public key:

matrix G over $\mathbf{F}_2$.

Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$.

Uses secret codeword mG,

weight-w error vector e.

McTiny:
McEliece for tiny network servers

Daniel J. Bernstein,
`uic.edu`, `rub.de`

Tanja Lange, `tue.nl`

Fundamental literature:
1962 Prange (attack)
+ many more attack papers.
1968 Berlekamp (decoder).
1970–1971 Goppa (codes).
1978 McEliece (cryptosystem).
1986 Niederreiter (compression)
+ many more optimizations.

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$.
Uses secret codeword mG,
weight-w error vector e.

1978 parameters for 2^{64} security
goal: 512×1024 matrix, $w = 50$.

McTiny:
McEliece for tiny network servers

Daniel J. Bernstein,
uic.edu, rub.de

Tanja Lange, tue.nl

Fundamental literature:
1962 Prange (attack)
+ many more attack papers.
1968 Berlekamp (decoder).
1970–1971 Goppa (codes).
1978 McEliece (cryptosystem).
1986 Niederreiter (compression)
+ many more optimizations.

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$.
Uses secret codeword mG,
weight-w error vector e.

1978 parameters for 2^{64} security
goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated
with binary Goppa code structure
that allows efficient decoding:
$C \mapsto mG, e$.

e for tiny network servers

. Bernstein,

u, `rub.de`

ange, `tue.nl`

---

ental literature:

ange (attack)

more attack papers.

rlekamp (decoder).

71 Goppa (codes).

cEliece (cryptosystem).

ederreiter (compression)

more optimizations.

Encoding and decoding

1978 McEliece public key: matrix G over $\mathbf{F}_2$. Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$. Uses secret codeword mG, weight-w error vector e.

1978 parameters for 2^{64} security goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated with binary Goppa code structure that allows efficient decoding: $C \mapsto mG, e$.

Binary G

Paramet

$w \in \{2,$

$n \in \{w$ l

network servers

n,

.nl

ature:

ck)

ack papers.

decoder).

(codes).

yptosystem).

(compression)

imizations.

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$.
Uses secret codeword mG,
weight-w error vector e.

1978 parameters for 2^{64} security
goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated
with binary Goppa code structure
that allows efficient decoding:
$C \mapsto mG, e$.

Binary Goppa cod

Parameters: $q \in \{$
$w \in \{2, 3, \ldots, \lfloor(q$
$n \in \{w \lg q + 1, \ldots$

ervers

.

m).
ion)
.

Encoding and decoding

1978 McEliece public key: matrix G over $\mathbf{F}_2$. Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$. Uses secret codeword mG, weight-w error vector e.

1978 parameters for 2^{64} security goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated with binary Goppa code structure that allows efficient decoding: $C \mapsto mG, e$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots$
$w \in \{2, 3, \ldots, \lfloor (q-1)/ \lg q$
$n \in \{w \lg q + 1, \ldots, q - 1, q$

Encoding and decoding

1978 McEliece public key:
matrix G over $\mathbf{F}_2$.
Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$.
Uses secret codeword mG,
weight-w error vector e.

1978 parameters for 2^{64} security goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated with binary Goppa code structure that allows efficient decoding:
$C \mapsto mG, e$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Encoding and decoding

1978 McEliece public key: matrix G over $\mathbf{F}_2$. Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$. Uses secret codeword mG, weight-w error vector e.

1978 parameters for 2^{64} security goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated with binary Goppa code structure that allows efficient decoding: $C \mapsto mG, e$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$; $w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$; $n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$; monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

Encoding and decoding

1978 McEliece public key: matrix G over $\mathbf{F}_2$. Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$. Uses secret codeword mG, weight-w error vector e.

1978 parameters for 2^{64} security goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated with binary Goppa code structure that allows efficient decoding: $C \mapsto mG, e$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$; $w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$; $n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$; monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of the map $v \mapsto \sum_i v_i/(x - \alpha_i)$ from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$. Normally dimension $n - w \lg q$.

Encoding and decoding

1978 McEliece public key: matrix G over $\mathbf{F}_2$. Normally $m \mapsto mG$ is injective.

Ciphertext: vector $C = mG + e$. Uses secret codeword mG, weight-w error vector e.

1978 parameters for 2^{64} security goal: 512×1024 matrix, $w = 50$.

Public key is secretly generated with binary Goppa code structure that allows efficient decoding: $C \mapsto mG, e$.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$; $w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$; $n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$; monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of the map $v \mapsto \sum_i v_i/(x - \alpha_i)$ from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$. Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$ whose image is this code.

g and decoding

cEliece public key:

G over $\mathbf{F}_2$.

y $m \mapsto mG$ is injective.

xt: vector $C = mG + e$.

cret codeword mG,

w error vector e.

rameters for 2^{64} security

2 $\times$ 1024 matrix, $w = 50$.

ey is secretly generated

ary Goppa code structure

ows efficient decoding:

G, e.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x - \alpha_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$
whose image is this code.

One-way

Fundame

Can atta

random

key G ar

oding

blic key:

G is injective.

$C = mG + e$.

ord mG,

ctor e.

or 2^{64} security

matrix, $w = 50$.

tly generated

code structure

nt decoding:

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x - \alpha_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$
whose image is this code.

One-wayness (“OV

Fundamental secu
Can attacker effici
random m, e giver
key G and ciphert

ve.

$+ e$.

urity

$= 50$.

ted

ıcture

g:

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x - \alpha_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$
whose image is this code.

One-wayness ("OW-Passive"

Fundamental security questi
Can attacker efficiently find
random m, e given random p
key G and ciphertext $mG+e$

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x - \alpha_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$
whose image is this code.

One-wayness ("OW-Passive")

Fundamental security question:
Can attacker efficiently find
random m, e given random public
key G and ciphertext $mG + e$?

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w
polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of
the map $v \mapsto \sum_i v_i/(x-\alpha_i)$
from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$
whose image is this code.

One-wayness ("OW-Passive")

Fundamental security question:
Can attacker efficiently find
random m, e given random public
key G and ciphertext $mG+e$?

1962 Prange: simple attack idea
guiding sizes in 1978 McEliece.

Binary Goppa codes

Parameters: $q \in \{8, 16, 32, \ldots\}$;
$w \in \{2, 3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
$n \in \{w \lg q + 1, \ldots, q-1, q\}$.

Secrets: distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
monic irreducible degree-w polynomial $g \in \mathbf{F}_q[x]$.

Goppa code: kernel of the map $v \mapsto \sum_i v_i/(x - \alpha_i)$ from $\mathbf{F}_2^n$ to $\mathbf{F}_q[x]/g$.
Normally dimension $n - w \lg q$.

McEliece uses random $G \in \mathbf{F}_2^{k \times n}$ whose image is this code.

One-wayness ("OW-Passive")

Fundamental security question: Can attacker efficiently find random m, e given random public key G and ciphertext $mG+e$?

1962 Prange: simple attack idea guiding sizes in 1978 McEliece.

The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against Prange's attack.
Here $c_0 \approx 0.7418860694$.

Goppa codes

ers: $q \in \{8, 16, 32, \ldots\}$;
$3, \ldots, \lfloor (q-1)/\lg q \rfloor\}$;
g $q+1, \ldots, q-1, q\}$.

distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;
reducible degree-w
ial $g \in \mathbf{F}_q[x]$.

ode: kernel of
$v \mapsto \sum_i v_i/(x - \alpha_i)$
to $\mathbf{F}_q[x]/g$.
y dimension $n - w \lg q$.

e uses random $G \in \mathbf{F}_2^{k \times n}$
mage is this code.

One-wayness ("OW-Passive")

Fundamental security question:
Can attacker efficiently find
random m, e given random public
key G and ciphertext $mG+e$?

1962 Prange: simple attack idea
guiding sizes in 1978 McEliece.

The McEliece system
(with later key-size optimizations)
uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys
as $\lambda \to \infty$ to achieve 2^λ security
against Prange's attack.
Here $c_0 \approx 0.7418860694$.

≥26 sub
analyzin

1981 Cla
cre
1988 Le
1988 Le
1989 Kr
1989 Ste
1989 Du
1990 Co
1990 va
1991 Du
1991 Co
1993 Ch
1993 Ch

es

$8, 16, 32, \ldots\}$;

$-1)/\lg q\rfloor\}$;

$., q-1, q\}$.

$\alpha_1, \ldots, \alpha_n \in \mathbf{F}_q$;

degree-w

$_q[x]$.

el of

$v_i/(x-\alpha_i)$

g.

on $n - w \lg q$.

dom $G \in \mathbf{F}_2^{k\times n}$

is code.

One-wayness ("OW-Passive")

Fundamental security question: Can attacker efficiently find random m, e given random public key G and ciphertext $mG+e$?

1962 Prange: simple attack idea guiding sizes in 1978 McEliece.

The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against Prange's attack. Here $c_0 \approx 0.7418860694$.

$\geq$26 subsequent p

analyzing one-way

1981 Clark–Cain,

crediting Om

1988 Lee–Brickell.

1988 Leon.

1989 Krouk.

1989 Stern.

1989 Dumer.

1990 Coffey–Good

1990 van Tilburg.

1991 Dumer.

1991 Coffey–Good

1993 Chabanne–C

1993 Chabaud.

. . .};

⌋};

q}.

$\in \mathbf{F}_q$;

)

g q.

$\mathbf{F}_2^{k\times n}$

One-wayness ("OW-Passive")

Fundamental security question: Can attacker efficiently find random m, e given random public key G and ciphertext $mG+e$?

1962 Prange: simple attack idea guiding sizes in 1978 McEliece.

The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against Prange's attack. Here $c_0 \approx 0.7418860694$.

$\geq$26 subsequent publication
analyzing one-wayness of sy

1981 Clark–Cain, crediting Omura.
1988 Lee–Brickell.
1988 Leon.
1989 Krouk.
1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg.
1991 Dumer.
1991 Coffey–Goodman–Farr
1993 Chabanne–Courteau.
1993 Chabaud.

One-wayness ("OW-Passive")

Fundamental security question:
Can attacker efficiently find random m, e given random public key G and ciphertext $mG+e$?

1962 Prange: simple attack idea guiding sizes in 1978 McEliece.

The McEliece system (with later key-size optimizations) uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against Prange's attack.
Here $c_0 \approx 0.7418860694$.

$\geq$26 subsequent publications analyzing one-wayness of system:

1981 Clark–Cain, crediting Omura.
1988 Lee–Brickell.
1988 Leon.
1989 Krouk.
1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg.
1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.

ness ("OW-Passive")

ental security question:
acker efficiently find
m, e given random public
nd ciphertext $mG+e$?

ange: simple attack idea
sizes in 1978 McEliece.

Eliece system
ter key-size optimizations)
$+ o(1))\lambda^2(\lg\lambda)^2$-bit keys
∞ to achieve 2^λ security
Prange's attack.
≈ 0.7418860694.

$\geq$26 subsequent publications
analyzing one-wayness of system:

1981 Clark–Cain,
crediting Omura.
1988 Lee–Brickell.
1988 Leon.
1989 Krouk.
1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg.
1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.

1994 var
1994 Ca
1998 Ca
1998 Ca
2008 Be
2009 Be
var
2009 Fir
2011 Be
2011 Ma
2012 Be
2013 Ha
2015 Ma
2016 Ca
2017 Bo

W-Passive")

rity question:
ently find
n random public
ext $mG+e$?

ple attack idea
978 McEliece.

em
e optimizations)
$^2(\lg\lambda)^2$-bit keys
eve 2^λ security
ttack.
360694.

≥26 subsequent publications
analyzing one-wayness of system:

1981 Clark–Cain,
crediting Omura.
1988 Lee–Brickell.
1988 Leon.
1989 Krouk.
1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg.
1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.

1994 van Tilburg.
1994 Canteaut–Ch
1998 Canteaut–Ch
1998 Canteaut–Se
2008 Bernstein–La
2009 Bernstein–La
van Tilborg.
2009 Finiasz–Send
2011 Bernstein–La
2011 May–Meurer
2012 Becker–Joux
2013 Hamdaoui–S
2015 May–Ozerov
2016 Canto Torres
2017 Both–May.

’)

on:

public
e?

idea
ece.

tions)
t keys
curity

≥26 subsequent publications
analyzing one-wayness of system:

1981 Clark–Cain,
crediting Omura.
1988 Lee–Brickell.
1988 Leon.
1989 Krouk.
1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg.
1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peter
2009 Bernstein–Lange–Peter
van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peter
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Me
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier
2017 Both–May.

$\geq$26 subsequent publications analyzing one-wayness of system:

1981 Clark–Cain,
crediting Omura.
1988 Lee–Brickell.
1988 Leon.
1989 Krouk.
1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg.
1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–Peters–
van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.
2017 Both–May.

osequent publications
g one-wayness of system:

ark–Cain,
editing Omura.
e–Brickell.
on.
ouk.
ern.
umer.
ffey–Goodman.
n Tilburg.
umer.
ffey–Goodman–Farrell.
abanne–Courteau.
abaud.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.
2017 Both–May.

The McI
uses (c_0
as $\lambda \rightarrow$
against a
Same c_0

ublications
ness of system:

ura.

lman.

lman–Farrell.
ourteau.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.
2017 Both–May.

The McEliece syst
uses $(c_0 + o(1))\lambda^2$
as $\lambda \to \infty$ to achi
against all attacks
Same $c_0 \approx 0.7418$

s

stem:

ell.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peters.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Finiasz–Sendrier.

2011 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2012 Becker–Joux–May–Meurer.

2013 Hamdaoui–Sendrier.

2015 May–Ozerov.

2016 Canto Torres–Sendrier.

2017 Both–May.

The McEliece system
uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bi
as $\lambda \to \infty$ to achieve 2^λ sec
against all attacks known to
Same $c_0 \approx 0.7418860694$.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.
2017 Both–May.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.
2017 Both–May.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Finiasz–Sendrier.
2011 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2012 Becker–Joux–May–Meurer.
2013 Hamdaoui–Sendrier.
2015 May–Ozerov.
2016 Canto Torres–Sendrier.
2017 Both–May.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

Modern example, `mceliece6960119` parameter set (2008 Bernstein–Lange–Peters): $q = 8192$, $n = 6960$, $w = 119$.

n Tilburg.
nteaut–Chabanne.
nteaut–Chabaud.
nteaut–Sendrier.
rnstein–Lange–Peters.
rnstein–Lange–Peters–
n Tilborg.
niasz–Sendrier.
rnstein–Lange–Peters.
ay–Meurer–Thomae.
cker–Joux–May–Meurer.
mdaoui–Sendrier.
ay–Ozerov.
nto Torres–Sendrier.
th–May.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

Modern example, `mceliece6960119` parameter set (2008 Bernstein–Lange–Peters): $q = 8192$, $n = 6960$, $w = 119$.

NIST co

2016: U
Standar
"post-qu

2017: 6

2019: N
26 subm

habanne.
habaud.
endrier.
ange–Peters.
ange–Peters–

drier.
ange–Peters.
–Thomae.
–May–Meurer.
endrier.
.
s–Sendrier.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

Modern example, `mceliece6960119` parameter set (2008 Bernstein–Lange–Peters): $q = 8192$, $n = 6960$, $w = 119$.

NIST competition

2016: U.S. Nation
Standards and Tec
“post-quantum” c

2017: 69 complete

2019: NIST select
26 submissions for

rs.

rs–

rs.

urer.

.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

Modern example, `mceliece6960119` parameter set (2008 Bernstein–Lange–Peters): $q = 8192$, $n = 6960$, $w = 119$.

NIST competition

2016: U.S. National Institut
Standards and Technology s
"post-quantum" competitio

2017: 69 complete submissi

2019: NIST selects
26 submissions for round 2.

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today. Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks: 2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

Modern example, `mceliece6960119` parameter set (2008 Bernstein–Lange–Peters): $q = 8192$, $n = 6960$, $w = 119$.

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects 26 submissions for round 2.

The McEliece system
uses $(c_0 + o(1))\lambda^2(\lg\lambda)^2$-bit keys
as $\lambda \to \infty$ to achieve 2^λ security
against all attacks known today.
Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ
stops all known *quantum* attacks:
2008 Bernstein, 2017 Kachigar–Tillich, 2018 Kirshanova.

Modern example,
`mceliece6960119` parameter set
(2008 Bernstein–Lange–Peters):
$q = 8192$, $n = 6960$, $w = 119$.

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects
26 submissions for round 2.

"Classic McEliece": submission
from team of 12 people.

Round-2 options:
8192128, 6960119, 6688128,
460896, 348864.

Eliece system

$+ o(1))\lambda^2(\lg \lambda)^2$-bit keys

∞ to achieve 2^λ security

all attacks known today.

≈ 0.7418860694.

g λ with 2λ

known *quantum* attacks:

rnstein, 2017 Kachigar–

2018 Kirshanova.

example,

ce6960119 parameter set

ernstein–Lange–Peters):

2, $n = 6960$, $w = 119$.

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects
26 submissions for round 2.

"Classic McEliece": submission from team of 12 people.

Round-2 options:
8192128, 6960119, 6688128,
460896, 348864.

Is Classi

1978 Mc

1978 Mc

huge am

Some wo

while cle

e.g., Nie

e.g., ma

Classic M

Classic M

more tha

em
$^2(\lg\lambda)^2$-bit keys
eve 2^λ security
known today.
860694.

2λ
uantum attacks:
017 Kachigar–
anova.

9 parameter set
ange–Peters):
60, $w = 119$.

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects 26 submissions for round 2.

"Classic McEliece": submission from team of 12 people.

Round-2 options:
8192128, 6960119, 6688128, 460896, 348864.

Is Classic McEliec
1978 McEliece? N

1978 McEliece pro
huge amount of fo

Some work improv
while clearly prese
e.g., Niederreiter c
e.g., many decodin
Classic McEliece u

Classic McEliece a
more than OW-Pa

t keys
curity
oday.

tacks:
gar–

er set
ers):
19.

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects
26 submissions for round 2.

"Classic McEliece": submission from team of 12 people.

Round-2 options:
8192128, 6960119, 6688128,
460896, 348864.

Is Classic McEliece same as
1978 McEliece? Not exactly

1978 McEliece prompted a
huge amount of followup wo

Some work improves efficien
while clearly preserving secu
e.g., Niederreiter compressio
e.g., many decoding speedu
Classic McEliece uses all thi

Classic McEliece also aims f
more than OW-Passive secu

NIST competition

2016: U.S. National Institute of Standards and Technology starts "post-quantum" competition.

2017: 69 complete submissions.

2019: NIST selects 26 submissions for round 2.

"Classic McEliece": submission from team of 12 people.

Round-2 options:
8192128, 6960119, 6688128, 460896, 348864.

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter compression; e.g., many decoding speedups. Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

mpetition

.S. National Institute of ds and Technology starts uantum" competition.

9 complete submissions.

IST selects issions for round 2.

McEliece": submission m of 12 people.

options:
3, 6960119, 6688128,
348864.

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security:
e.g., Niederreiter compression;
e.g., many decoding speedups.
Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

Niederre

Generat of length $G' \in \mathbf{F}_2^{k}$

McEliec random

nal Institute of
chnology starts
ompetition.

e submissions.

s
round 2.

": submission
eople.

9, 6688128,

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter compression; e.g., many decoding speedups. Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

Niederreiter key co

Generator matrix f
of length n and di
$G' \in \mathbf{F}_2^{k \times n}$ with Γ

McEliece public ke
random invertible

e of
tarts
n.

ons.

sion

8,

Is Classic McEliece same as
1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter compression; e.g., many decoding speedups. Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

Niederreiter key compressio

Generator matrix for code Γ
of length n and dimension k
$G' \in \mathbf{F}_2^{k \times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = S$
random invertible $S \in \mathbf{F}_2^{k \times k}$

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter compression; e.g., many decoding speedups. Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter compression; e.g., many decoding speedups. Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Is Classic McEliece same as 1978 McEliece? Not exactly.

1978 McEliece prompted a huge amount of followup work.

Some work improves efficiency while clearly preserving security: e.g., Niederreiter compression; e.g., many decoding speedups. Classic McEliece uses all this.

Classic McEliece also aims for more than OW-Passive security.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k \times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k \times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k | R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

c McEliece same as
cEliece? Not exactly.

cEliece prompted a
nount of followup work.

ork improves efficiency
early preserving security:
ederreiter compression;
ny decoding speedups.
McEliece uses all this.

McEliece also aims for
an OW-Passive security.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k \times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k \times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederre

Use Nied

McEliece

e same as
lot exactly.

ompted a
ollowup work.

ves efficiency
rving security:
compression;
ng speedups.
uses all this.

also aims for
ssive security.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederreiter cipher

Use Niederreiter k

McEliece ciphertex

y.

ork.

cy

rity:

on;

ps.

s.

or

rity.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k \times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k \times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederreiter ciphertext comp

Use Niederreiter key $G = (I_$

McEliece ciphertext: $mG + e$

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: <2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter: $He^\top \in \mathbf{F}_2^{(n-k)\times 1}$ where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$, can attacker efficiently find e?

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k \times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k \times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k | R)$.

Pr $\approx$29% that systematic form exists. Security loss: <2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k | R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter: $He^\top \in \mathbf{F}_2^{(n-k) \times 1}$ where $H = (R^\top | I_{n-k})$.

Given H and Niederreiter's $He^\top$, can attacker efficiently find e?

If so, attacker can efficiently find m, e given G and $mG + e$:

Niederreiter key compression

Generator matrix for code Γ of length n and dimension k: $G' \in \mathbf{F}_2^{k\times n}$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

McEliece public key: $G = SG'$ for random invertible $S \in \mathbf{F}_2^{k\times k}$.

Niederreiter instead reduces G' to the unique generator matrix in systematic form: $G = (I_k|R)$.

Pr $\approx$29% that systematic form exists. Security loss: $<$2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG+e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter: $He^\top \in \mathbf{F}_2^{(n-k)\times 1}$ where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$, can attacker efficiently find e?

If so, attacker can efficiently find m, e given G and $mG + e$: compute $H(mG + e)^\top = He^\top$; find e; compute m from mG.

iter key compression

or matrix for code Γ

n n and dimension k:

$\times n$ with $\Gamma = \mathbf{F}_2^k \cdot G'$.

e public key: $G = SG'$ for

invertible $S \in \mathbf{F}_2^{k \times k}$.

iter instead reduces G'

nique generator matrix in

tic form: $G = (I_k|R)$.

% that systematic form

Security loss: <2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$,
can attacker efficiently find e?

If so, attacker can efficiently
find m, e given G and $mG + e$:
compute $H(mG + e)^\top = He^\top$;
find e; compute m from mG.

Other ch

Niederre

Solomon

by Sidel

More co

codes, R

AG code

several L

ompression

for code Γ

mension k:

$= \mathbf{F}_2^k \cdot G'$.

ey: $G = SG'$ for

$S \in \mathbf{F}_2^{k\times k}$.

d reduces G'

erator matrix in

$G = (I_k|R)$.

tematic form

ss: <2 bits.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$,
can attacker efficiently find e?

If so, attacker can efficiently
find m, e given G and $mG + e$:
compute $H(mG + e)^\top = He^\top$;
find e; compute m from mG.

Other choices of c

Niederreiter sugge

Solomon codes. B

by Sidelnikov and

More corpses: e.g.

codes, Reed–Mulle

AG codes, Gabidu

several LDPC code

n

r:

G' for

.

G'

rix in

).

rm

s.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$,
can attacker efficiently find e?

If so, attacker can efficiently
find m, e given G and $mG + e$:
compute $H(mG + e)^\top = He^\top$;
find e; compute m from mG.

Other choices of codes

Niederreiter suggested Reed-
Solomon codes. Broken in 1
by Sidelnikov and Shestakov

More corpses: e.g., concaten
codes, Reed–Muller codes, s
AG codes, Gabidulin codes,
several LDPC codes.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG + e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$, can attacker efficiently find e?

If so, attacker can efficiently find m, e given G and $mG + e$: compute $H(mG + e)^\top = He^\top$; find e; compute m from mG.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

Niederreiter ciphertext compression

Use Niederreiter key $G = (I_k|R)$.

McEliece ciphertext: $mG+e \in \mathbf{F}_2^n$.

Niederreiter ciphertext, shorter:
$He^\top \in \mathbf{F}_2^{(n-k)\times 1}$
where $H = (R^\top|I_{n-k})$.

Given H and Niederreiter's $He^\top$, can attacker efficiently find e?

If so, attacker can efficiently find m, e given G and $mG + e$: compute $H(mG + e)^\top = He^\top$; find e; compute m from mG.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

eiter ciphertext compression

derreiter key $G = (I_k|R)$.

e ciphertext: $mG+e \in \mathbf{F}_2^n$.

eiter ciphertext, shorter:
$\mathbf{F}_2^{(n-k)\times 1}$
$= (R^\top|I_{n-k})$.

and Niederreiter's $He^\top$,
cker efficiently find e?

tacker can efficiently
e given G and $mG+e$:
e $H(mG+e)^\top = He^\top$;
ompute m from mG.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

IND-CC

OW-Pas
Message
Attacker
and obse

rtext compression

ey $G = (I_k|R)$.

xt: $mG+e \in \mathbf{F}_2^n$.

rtext, shorter:

$_{n-k})$.

erreiter's $He^\top$,

ently find e?

efficiently

and $mG + e$:

$e)^\top = He^\top$;

from mG.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

IND-CCA2 securit

OW-Passive securi

Messages are not

Attackers choose c

and observe reacti

pression

$_k|R)$.

$e \in \mathbf{F}_2^n$.

ter:

$He^\top$,

e?

y

$+ e$:

$e^\top$;

S.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

IND-CCA2 security

OW-Passive security is too w

Messages are not random.

Attackers choose ciphertexts

and observe reactions.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

IND-CCA2 security

OW-Passive security is too weak.
Messages are not random.
Attackers choose ciphertexts and observe reactions.

Other choices of codes

Niederreiter suggested Reed–Solomon codes. Broken in 1992 by Sidelnikov and Shestakov.

More corpses: e.g., concatenated codes, Reed–Muller codes, several AG codes, Gabidulin codes, several LDPC codes.

No proof that changing codes preserves security level.

Classic McEliece: binary Goppa.

IND-CCA2 security

OW-Passive security is too weak. Messages are not random. Attackers choose ciphertexts and observe reactions.

Classic McEliece does more work for "IND-CCA2 security".

Combines coding theory with AES-GCM "authenticated cipher" and SHA-3 "hash function".

All messages are safe. Reusing keys is safe.

hoices of codes

eiter suggested Reed–
n codes. Broken in 1992
nikov and Shestakov.

rpses: e.g., concatenated
Reed–Muller codes, several
es, Gabidulin codes,
LDPC codes.

f that changing codes
s security level.

McEliece: binary Goppa.

IND-CCA2 security

OW-Passive security is too weak.
Messages are not random.
Attackers choose ciphertexts
and observe reactions.

Classic McEliece does more work
for "IND-CCA2 security".

Combines coding theory with
AES-GCM "authenticated cipher"
and SHA-3 "hash function".

All messages are safe.
Reusing keys is safe.

Time

Cycles o

params
348864
460896
6688128
6960119
8192128
348864
460896
6688128
6960119
8192128

odes

sted Reed–
roken in 1992
Shestakov.

., concatenated
er codes, several
lin codes,
es.

nging codes
level.

binary Goppa.

IND-CCA2 security

OW-Passive security is too weak.
Messages are not random.
Attackers choose ciphertexts
and observe reactions.

Classic McEliece does more work
for "IND-CCA2 security".

Combines coding theory with
AES-GCM "authenticated cipher"
and SHA-3 "hash function".

All messages are safe.
Reusing keys is safe.

Time

Cycles on Intel Ha

params	op	cy
348864	enc	45
460896	enc	82
6688128	enc	153
6960119	enc	154
8192128	enc	183
348864	dec	136
460896	dec	273
6688128	dec	320
6960119	dec	302
8192128	dec	324

–
.992
.
nated
everal
es
ppa.

IND-CCA2 security

OW-Passive security is too weak.
Messages are not random.
Attackers choose ciphertexts
and observe reactions.

Classic McEliece does more work
for “IND-CCA2 security”.

Combines coding theory with
AES-GCM “authenticated cipher”
and SHA-3 “hash function”.

All messages are safe.
Reusing keys is safe.

Time

Cycles on Intel Haswell CPU

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

IND-CCA2 security

OW-Passive security is too weak. Messages are not random. Attackers choose ciphertexts and observe reactions.

Classic McEliece does more work for "IND-CCA2 security".

Combines coding theory with AES-GCM "authenticated cipher" and SHA-3 "hash function".

All messages are safe. Reusing keys is safe.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

A2 security

sive security is too weak.
es are not random.
rs choose ciphertexts
erve reactions.

McEliece does more work
D-CCA2 security".

es coding theory with
M "authenticated cipher"
A-3 "hash function".

sages are safe.
keys is safe.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, y
most im
to have

params
348864
348864f
460896
460896f
668812
668812
696011
696011
819212
819212

y

ity is too weak.
random.
ciphertexts
ons.

loes more work
curity".

theory with
nticated cipher"
function".

afe.
fe.

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, you're leav
most important co
to have such slow

params	op
348864	keyger
348864f	keyger
460896	keyger
460896f	keyger
6688128	keyger
6688128f	keyger
6960119	keyger
6960119f	keyger
8192128	keyger
8192128f	keyger

weak.

s

work

h

ipher"

-

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, you're leaving out th
most important cost! It's cr
to have such slow keygen!"

params	op	cy
348864	keygen	140870
348864f	keygen	82232
460896	keygen	441517
460896f	keygen	282869
6688128	keygen	1180468
6688128f	keygen	625470
6960119	keygen	1109340
6960119f	keygen	564570
8192128	keygen	933422
8192128f	keygen	678860

Time

Cycles on Intel Haswell CPU core:

params	op	cycles
348864	enc	45888
460896	enc	82684
6688128	enc	153372
6960119	enc	154972
8192128	enc	183892
348864	dec	136840
460896	dec	273872
6688128	dec	320428
6960119	dec	302460
8192128	dec	324008

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

n Intel Haswell CPU core:

	op	cycles
	enc	45888
	enc	82684
3	enc	153372
9	enc	154972
3	enc	183892
	dec	136840
	dec	273872
3	dec	320428
9	dec	302460
3	dec	324008

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What

that this

a proble

swell CPU core:

ycles
5888
2684
3372
4972
3892
5840
3872
0428
2460
4008

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence
that this keygen ti
a problem for appl

J core:

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we hav that this keygen time is a problem for applications?

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

"Wait, you're leaving out the most important cost! It's crazy to have such slow keygen!"

params	op	cycles
348864	keygen	140870324
348864f	keygen	82232360
460896	keygen	441517292
460896f	keygen	282869316
6688128	keygen	1180468912
6688128f	keygen	625470504
6960119	keygen	1109340668
6960119f	keygen	564570384
8192128	keygen	933422948
8192128f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

ou're leaving out the
portant cost! It's crazy
such slow keygen!"

	op	cycles
	keygen	140870324
f	keygen	82232360
	keygen	441517292
f	keygen	282869316
3	keygen	1180468912
3f	keygen	625470504
9	keygen	1109340668
9f	keygen	564570384
3	keygen	933422948
3f	keygen	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes co

params
348864
460896
668812
696011
819212
348864
460896
668812
696011
819212

"It's cra

ing out the
ost! It's crazy
keygen!"

	cycles
n	140870324
n	82232360
n	441517292
n	282869316
n	1180468912
n	625470504
n	1109340668
n	564570384
n	933422948
n	678860388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes communicat

params	object
348864	cipherte
460896	cipherte
6688128	cipherte
6960119	cipherte
8192128	cipherte
348864	key
460896	key
6688128	key
6960119	key
8192128	key

"It's crazy to have

e
azy

cles

0324
2360
7292
0316
3912
0504
0668
0384
2948
0388

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes communicated

params	object	byte
348864	ciphertext	12
460896	ciphertext	18
6688128	ciphertext	24
6960119	ciphertext	22
8192128	ciphertext	24
348864	key	26112
460896	key	52416
6688128	key	104499
6960119	key	104731
8192128	key	135782

“It's crazy to have big keys!

1. What evidence do we have that this keygen time is a problem for applications?

2. Classic McEliece is designed for IND-CCA2 security, so a key can be generated once and used a huge number of times.

3. McEliece's binary operations are very well suited for hardware. See 2018 Wang–Szefer–Niederhagen. Isn't this what's most important for the future?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

t evidence do we have
s keygen time is
m for applications?

ic McEliece is designed
-CCA2 security, so
n be generated once and
uge number of times.

iece's binary operations
well suited for hardware.
8 Wang–Szefer–
agen. Isn't this what's
portant for the future?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What ev
that the
a proble

do we have
me is
lications?

ce is designed
urity, so
rated once and
per of times.

ary operations
d for hardware.
zefer–
t this what's
r the future?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do
that these key size
a problem for appl

ve

ned

e and
s.

ons
ware.

t's
re?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Bytes communicated

params	object	bytes
348864	ciphertext	128
460896	ciphertext	188
6688128	ciphertext	240
6960119	ciphertext	226
8192128	ciphertext	240
348864	key	261120
460896	key	524160
6688128	key	1044992
6960119	key	1047319
8192128	key	1357824

"It's crazy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

ommunicated

	object	bytes
	ciphertext	128
	ciphertext	188
3	ciphertext	240
9	ciphertext	226
3	ciphertext	240
	key	261120
	key	524160
3	key	1044992
9	key	1047319
3	key	1357824

zy to have big keys!"

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 Mc
postquai
Use star
techniqu
etc.) to
commun

Each cip
the way
the serve
can ofte
much fa

Again IN

ted

	bytes
xt	128
xt	188
xt	240
xt	226
xt	240
	261120
	524160
	1044992
	1047319
	1357824

e big keys!"

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 McGrew "Liv
postquantum cryp
Use standard netw
techniques (multic
etc.) to reduce cos
communicating pu

Each ciphertext ha
the way between t
the server, but pul
can often be retrie
much faster local

Again IND-CCA2

es

28
38
40
26
40

20
60
92
19
24
”

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 McGrew “Living with postquantum cryptography”
Use standard networking techniques (multicasts, cach
etc.) to reduce cost of communicating public keys.

Each ciphertext has to trave
the way between the client a
the server, but public keys
can often be retrieved throu
much faster local network.

Again IND-CCA2 is critical.

What evidence do we have that these key sizes are a problem for applications?

Compare to, e.g., web-page size.

`httparchive.org` statistics:
50% of web pages are >1.8MB.
25% of web pages are >3.5MB.
10% of web pages are >6.5MB.
The sizes keep growing.

Typically browser receives one web page from multiple servers, but reuses servers for more pages.
Is key size a big part of this?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

vidence do we have
se key sizes are
m for applications?

e to, e.g., web-page size.

chive.org statistics:
web pages are >1.8MB.
web pages are >3.5MB.
web pages are >6.5MB.
s keep growing.

y browser receives one web
m multiple servers, but
ervers for more pages.
ze a big part of this?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial o

Standard
strategy
of conne
up all m
for keepi

SYN flo

Server is
some co
connecti

we have
es are
lications?

web-page size.

g statistics:
are >1.8MB.
are >3.5MB.
are >6.5MB.
owing.

receives one web
e servers, but
more pages.
art of this?

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost
strategy: make a l
of connections to
up all memory ava
for keeping track o

SYN flood, HTTP

Server is forced to
some connections,
connections from l

size.

s:

MB.

MB.

MB.

ne web

but

s.

?

2015 McGrew “Living with postquantum cryptography”: Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge numb of connections to a server, f up all memory available on s for keeping track of connect

SYN flood, HTTP flood, etc

Server is forced to stop serv some connections, including connections from honest clie

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

2015 McGrew "Living with postquantum cryptography": Use standard networking techniques (multicasts, caching, etc.) to reduce cost of communicating public keys.

Each ciphertext has to travel all the way between the client and the server, but public keys can often be retrieved through much faster local network.

Again IND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

cGrew "Living with
ntum cryptography":
ndard networking
ues (multicasts, caching,
reduce cost of
nicating public keys.

phertext has to travel all
between the client and
er, but public keys
n be retrieved through
ster local network.

ND-CCA2 is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny n**
handles
each inc
without

ving with
tography":
orking
casts, caching,
st of
ublic keys.

as to travel all
the client and
blic keys
eved through
network.

is critical.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network se**
handles and imme
each incoming net
without allocating

:

ing,

el all
and

gh

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately for
each incoming network pack
without allocating any mem

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

Denial of service

Standard low-cost attack strategy: make a huge number of connections to a server, filling up all memory available on server for keeping track of connections.

SYN flood, HTTP flood, etc.

Server is forced to stop serving some connections, including connections from honest clients.

But some Internet protocols are *not* vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

f service

d low-cost attack
: make a huge number
ections to a server, filling
emory available on server
ing track of connections.

od, HTTP flood, etc.

s forced to stop serving
nnections, including
ons from honest clients.

ne Internet protocols
vulnerable to this attack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: “anonymous NFS”.

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an “input continuation” fits into a network packet.

“Here’s
McEliec

attack
huge number
a server, filling
ilable on server
of connections.

flood, etc.

stop serving
including
honest clients.

protocols
to this attack.

A **tiny network server**
handles and immediately forgets
each incoming network packet,
without allocating any memory.

Can use tiny network servers
to publish information.
Unauthenticated example from
last century: “anonymous NFS”.

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol
to use a tiny network server
if an “input continuation”
fits into a network packet.

“Here’s a natural s
McEliece can’t pos

per
illing
server
ions.
c.
ing
ents.
tack.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario th
McEliece can't possibly hand

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: “anonymous NFS”.

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an “input continuation” fits into a network packet.

“Here’s a natural scenario that McEliece can’t possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

A **tiny network server** handles and immediately forgets each incoming network packet, without allocating any memory.

Can use tiny network servers to publish information. Unauthenticated example from last century: "anonymous NFS".

1997 Aura–Nikander, 2005 Shieh–Myers–Sirer modify any protocol to use a tiny network server *if* an "input continuation" fits into a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

network server

and immediately forgets
oming network packet,
allocating any memory.

tiny network servers
sh information.
nticated example from
tury: "anonymous NFS".

ura–Nikander, 2005 Shieh–
Sirer modify any protocol
tiny network server
nput continuation"
a network packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstei
handles

erver

diately forgets
work packet,
any memory.

ork servers
tion.
example from
onymous NFS".

der, 2005 Shieh–
fy any protocol
ork server
nuation"
packet.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "
handles this scena

gets
et,
ory.

s

om
IFS”.

Shieh–
tocol

“Here’s a natural scenario that McEliece can’t possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It’s crazy if post-quantum standards can’t handle this!”

Bernstein–Lange “McTiny” handles this scenario.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

"Here's a natural scenario that McEliece can't possibly handle:

- To stop memory floods, I want a tiny network server.
- For forward secrecy, I want the server to encrypt a session key to an ephemeral public key sent by the client.
- This forces the public key to fit into a network packet. Is that 1500 bytes? Or 1280? Either way, your key is too big.

It's crazy if post-quantum standards can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything. Remaining problem: within this session, encrypt to an ephemeral key for forward secrecy.

a natural scenario that
e can't possibly handle:
op memory floods,
a tiny network server.
rward secrecy,
the server to encrypt a
n key to an ephemeral
key sent by the client.
orces the public key
nto a network packet.
t 1500 bytes? Or 1280?
way, your key is too big.
y if post-quantum
ls can't handle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.
Remaining problem:
within this session, encrypt to an ephemeral key for forward secrecy.

2. Clien
public ke

$$\begin{pmatrix} K_{1,1} \\ K_{2,1} \\ \vdots \\ K_{r,1} \end{pmatrix}$$

Each blo
to fit int

scenario that
ssibly handle:
floods,
twork server.
ecy,
r to encrypt a
n ephemeral
by the client.
public key
work packet.
es? Or 1280?
key is too big.
uantum
ndle this!"

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything. Remaining problem: within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decompo
public key $K = R$

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K \\ K_{2,1} & K_{2,2} & K \\ \vdots & \vdots & \\ K_{r,1} & K_{r,2} & K \end{pmatrix}$$

Each block is sma
to fit into a netwo

hat
dle:

er.

pt a
al
nt.

et.
80?
o big.

,

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.

Remaining problem: within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decomposes ephem
public key $K = R^\top$ into bloc

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \cdots \\ K_{2,1} & K_{2,2} & K_{2,3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ K_{r,1} & K_{r,2} & K_{r,3} & \cdots \end{pmatrix}$$

Each block is small enough to fit into a network packet.

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.

Remaining problem: within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

Bernstein–Lange "McTiny" handles this scenario.

1. The easy part: Client encrypts session key to server's long-term McEliece public key. This establishes an encrypted authenticated session.

Attacker who records this session and later steals server's secret key can then decrypt everything.
Remaining problem:
within this session, encrypt to an ephemeral key for forward secrecy.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server.
Server sends back $K_{i,j} e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client.
Key is erased after a few minutes.

n–Lange "McTiny"
this scenario.

easy part: Client
. session key to server's
m McEliece public key.
ablishes an encrypted
cated session.

r who records this session
r steals server's secret key
n decrypt everything.
ng problem:
his session, encrypt to an
ral key for forward secrecy.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j} e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Clien
containi
Server se

"McTiny"
rio.

Client
ey to server's
e public key.
n encrypted
ion.

rds this session
rver's secret key
everything.
m:
, encrypt to an
forward secrecy.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends on
containing several
Server sends back

er's
ey.
d

ession
et key
.

to an
ecrecy.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j} e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j} e_j^\top$. Server sends back combinati

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

2. Client decomposes ephemeral public key $K = R^\top$ into blocks:

$$\begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \dots & K_{1,\ell} \\ K_{2,1} & K_{2,2} & K_{2,3} & \dots & K_{2,\ell} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{r,1} & K_{r,2} & K_{r,3} & \dots & K_{r,\ell} \end{pmatrix}.$$

Each block is small enough to fit into a network packet.

3. Client sends $K_{i,j}$ to server. Server sends back $K_{i,j}e_j^\top$ encrypted to a server cookie key.

Server cookie key is not per-client. Key is erased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

Forward secrecy: Once cookie key and secret key for H are erased, client and server cannot decrypt.

t decomposes ephemeral
ey $K = R^\top$ into blocks:

$$\begin{matrix} K_{1,2} & K_{1,3} & \cdots & K_{1,\ell} \\ K_{2,2} & K_{2,3} & \cdots & K_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ K_{r,2} & K_{r,3} & \cdots & K_{r,\ell} \end{matrix}\Bigg).$$

ock is small enough
to a network packet.

t sends $K_{i,j}$ to server.
ends back $K_{i,j}e_j^\top$
ed to a server cookie key.

ookie key is not per-client.
rased after a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

Forward secrecy: Once cookie key and secret key for H are erased, client and server cannot decrypt.

Classic M

Security
by 40 ye

Cipherte

IND-CC

Open-so
fast cons
also FPG

No pate

Big keys
with tiny

https:/

oses ephemeral

$^\top$ into blocks:

$$\begin{matrix} {}_{1,3} & \cdots & K_{1,\ell} \\ {}_{2,3} & \cdots & K_{2,\ell} \\ \vdots & \ddots & \vdots \\ {}_{r,3} & \cdots & K_{r,\ell} \end{pmatrix}.$$

ll enough

rk packet.

$_{,j}$ to server.

$K_{i,j}e_j^\top$

ver cookie key.

is not per-client.

a few minutes.

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

Forward secrecy: Once cookie key and secret key for H are erased, client and server cannot decrypt.

Classic McEliece r

Security asymptot

by 40 years of cryp

Ciphertexts among

IND-CCA2 securit

Open-source imple

fast constant-time

also FPGA implem

No patents.

Big keys, but still

with tiny network

https://classic

neral

cks:

$$\begin{pmatrix} K_{1,\ell} \\ K_{2,\ell} \\ \vdots \\ K_{r,\ell} \end{pmatrix}.$$

er.

key.

-client.

nutes.

4. Client sends one packet containing several $K_{i,j} e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

Forward secrecy: Once cookie key and secret key for H are erased, client and server cannot decrypt.

Classic McEliece recap

Security asymptotics unchar

by 40 years of cryptanalysis.

Ciphertexts among the short

IND-CCA2 security.

Open-source implementatior

fast constant-time software,

also FPGA implementation.

No patents.

Big keys, but still compatibl

with tiny network servers.

https://classic.mcelie

4. Client sends one packet containing several $K_{i,j}e_j^\top$. Server sends back combination.

5. Repeat to combine everything, including I_{n-k} part of H.

6. Server sends final $He^\top$ directly to client, encrypted by session key but *not* by cookie key.

7. Client decrypts.

Forward secrecy: Once cookie key and secret key for H are erased, client and server cannot decrypt.

Classic McEliece recap

Security asymptotics unchanged by 40 years of cryptanalysis.

Ciphertexts among the shortest.

IND-CCA2 security.

Open-source implementations: fast constant-time software, also FPGA implementation.

No patents.

Big keys, but still compatible with tiny network servers.

https://classic.mceliece.org